Spring AOP Example

1. [Before Advice Example](http://www.javatpoint.com/spring-aop-example)
2. [After Returning Advice Example](http://www.javatpoint.com/spring-aop-example)
3. [Around Advice Example](http://www.javatpoint.com/spring-aop-example)
4. [After Throwing Advice Example](http://www.javatpoint.com/spring-aop-example)

There are given examples of **Spring1.2 old style AOP** (dtd based) implementation.

Though it is supported in spring 3, but it is recommended to use spring aop with aspectJ that we are going to learn in next page.

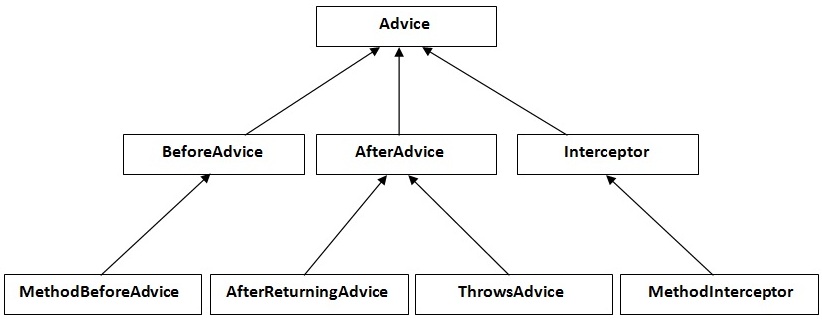
There are 4 types of advices supported in spring1.2 old style aop implementation.

1. **Before Advice** it is executed before the actual method call.
2. **After Advice** it is executed after the actual method call. If method returns a value, it is executed after returning value.
3. **Around Advice** it is executed before and after the actual method call.
4. **Throws Advice** it is executed if actual method throws exception.

***To understand the basic concepts of Spring AOP, visit the previous page.***

**Understanding the hierarchy of advice interfaces**

Let's understand the advice hierarchy by the diagram given below:



All are interfaces in aop.

**MethodBeforeAdvice** interface extends the **BeforeAdvice** interface.

**AfterReturningAdvice** interface extends the **AfterAdvice** interface.

**ThrowsAdvice** interface extends the **AfterAdvice** interface.

**MethodInterceptor** interface extends the **Interceptor** interface. It is used in around advice.

1) MethodBeforeAdvice Example

Create a class that contains actual business logic.

*File: A.java*

1. **package** com.javatpoint;
2. **public** **class** A {
3. **public** **void** m(){System.out.println("actual business logic");}
4. }

Now, create the advisor class that implements MethodBeforeAdvice interface.

*File: BeforeAdvisor.java*

1. **package** com.javatpoint;
2. **import** java.lang.reflect.Method;
3. **import** org.springframework.aop.MethodBeforeAdvice;
4. **public** **class** BeforeAdvisor **implements** MethodBeforeAdvice{
5. @Override
6. **public** **void** before(Method method, Object[] args, Object target)**throws** Throwable {
7. System.out.println("additional concern before actual logic");
8. }
9. }

In xml file, create 3 beans, one for A class, second for Advisor class and third for **ProxyFactoryBean** class.

*File: applicationContext.xml*

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="obj" **class**="com.javatpoint.A"></bean>
10. <bean id="ba" **class**="com.javatpoint.BeforeAdvisor"></bean>
12. <bean id="proxy" **class**="org.springframework.aop.framework.ProxyFactoryBean">
13. <property name="target" ref="obj"></property>
14. <property name="interceptorNames">
15. <list>
16. <value>ba</value>
17. </list>
18. </property>
19. </bean>
21. </beans>

**Understanding ProxyFactoryBean class:**

The **ProxyFactoryBean** class is provided by Spring Famework. It contains 2 properties target and interceptorNames. The instance of A class will be considered as target object and the instance of advisor class as interceptor. You need to pass the advisor object as the list object as in the xml file given above.

The ProxyFactoryBean class is written something like this:

1. **public** **class** ProxyFactoryBean{
2. **private** Object target;
3. **private** List interceptorNames;
4. //getters and setters
5. }

Now, let's call the actual method.

*File: Test.java*

1. **package** com.javatpoint;
2. **import** org.springframework.beans.factory.BeanFactory;
3. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
4. **import** org.springframework.core.io.ClassPathResource;
5. **import** org.springframework.core.io.Resource;
6. **public** **class** Test {
7. **public** **static** **void** main(String[] args) {
8. Resource r=**new** ClassPathResource("applicationContext.xml");
9. BeanFactory factory=**new** XmlBeanFactory(r);
11. A a=factory.getBean("proxy",A.**class**);
12. a.m();
13. }
14. }

**Output**

1. additional concern before actual logic
2. actual business logic

**Printing additional information in MethodBeforeAdvice**

We can print additional information like method name, method argument, target object, target object class name, proxy class etc.

You need to change only two classes BeforeAdvisor.java and Test.java.

*File: BeforeAdvisor.java*

1. **package** com.javatpoint;
2. **import** java.lang.reflect.Method;
3. **import** org.springframework.aop.MethodBeforeAdvice;
5. **public** **class** BeforeAdvisor **implements** MethodBeforeAdvice{
6. @Override
7. **public** **void** before(Method method, Object[] args, Object target)**throws** Throwable {
8. System.out.println("additional concern before actual logic");
9. System.out.println("method info:"+method.getName()+" "+method.getModifiers());
10. System.out.println("argument info:");
11. **for**(Object arg:args)
12. System.out.println(arg);
13. System.out.println("target Object:"+target);
14. System.out.println("target object class name: "+target.getClass().getName());
15. }
16. }

*File: Test.java*

1. **package** com.javatpoint;
2. **import** org.springframework.beans.factory.BeanFactory;
3. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
4. **import** org.springframework.core.io.ClassPathResource;
5. **import** org.springframework.core.io.Resource;
6. **public** **class** Test {
7. **public** **static** **void** main(String[] args) {
8. Resource r=**new** ClassPathResource("applicationContext.xml");
9. BeanFactory factory=**new** XmlBeanFactory(r);
11. A a=factory.getBean("proxy",A.**class**);
12. System.out.println("proxy class name: "+a.getClass().getName());
13. a.m();
14. }
15. }

**Output**

1. proxy **class** name: com.javatpoint.A$$EnhancerByCGLIB$$409872b1
2. additional concern before actual logic
3. method info:m 1
4. argument info:
5. target Object:com.javatpoint.A@11dba45
6. target object **class** name: com.javatpoint.A
7. actual business logic

2) AfterReturningAdvice Example

Create a class that contains actual business logic.

*File: A.java*

Same as in the previous example.

Now, create the advisor class that implements AfterReturningAdvice interface.

*File: AfterAdvisor.java*

1. **package** com.javatpoint;
2. **import** java.lang.reflect.Method;
3. **import** org.springframework.aop.AfterReturningAdvice;
4. **public** **class** AfterAdvisor **implements** AfterReturningAdvice{
5. @Override
6. **public** **void** afterReturning(Object returnValue, Method method,
7. Object[] args, Object target) **throws** Throwable {
9. System.out.println("additional concern after returning advice");
10. }
12. }

Create the xml file as in the previous example, you need to change only the advisor class here.

*File: applicationContext.xml*

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="obj" **class**="com.javatpoint.A"></bean>
10. <bean id="ba" **class**="com.javatpoint.AfterAdvisor"></bean>
12. <bean id="proxy" **class**="org.springframework.aop.framework.ProxyFactoryBean">
13. <property name="target" ref="obj"></property>
14. <property name="interceptorNames">
15. <list>
16. <value>ba</value>
17. </list>
18. </property>
19. </bean>
21. </beans>

*File: Test.java*

Same as in the previous example.

**Output**

1. actual business logic
2. additional concern after returning advice

3) MethodInterceptor (AroundAdvice) Example

Create a class that contains actual business logic.

*File: A.java*

Same as in the previous example.

Now, create the advisor class that implements MethodInterceptor interface.

*File: AroundAdvisor.java*

1. **package** com.javatpoint;
2. **import** org.aopalliance.intercept.MethodInterceptor;
3. **import** org.aopalliance.intercept.MethodInvocation;
4. **public** **class** AroundAdvisor **implements** MethodInterceptor{
6. @Override
7. **public** Object invoke(MethodInvocation mi) **throws** Throwable {
8. Object obj;
9. System.out.println("additional concern before actual logic");
10. obj=mi.proceed();
11. System.out.println("additional concern after actual logic");
12. **return** obj;
13. }
15. }

Create the xml file as in the previous example, you need to change only the advisor class here.

*File: applicationContext.xml*

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="obj" **class**="com.javatpoint.A"></bean>
10. <bean id="ba" **class**="com.javatpoint.AroundAdvisor"></bean>
12. <bean id="proxy" **class**="org.springframework.aop.framework.ProxyFactoryBean">
13. <property name="target" ref="obj"></property>
14. <property name="interceptorNames">
15. <list>
16. <value>ba</value>
17. </list>
18. </property>
19. </bean>
21. </beans>

*File: Test.java*

Same as in the previous example.

**Output**

1. additional concern before actual logic
2. actual business logic
3. additional concern after actual logic

4) ThrowsAdvice Example

Create a class that contains actual business logic.

*File: Validator.java*

1. **package** com.javatpoint;
2. **public** **class** Validator {
3. **public** **void** validate(**int** age)**throws** Exception{
4. **if**(age<18){
5. **throw** **new** ArithmeticException("Not Valid Age");
6. }
7. **else**{
8. System.out.println("vote confirmed");
9. }
10. }
11. }

Now, create the advisor class that implements ThrowsAdvice interface.

*File: ThrowsAdvisor.java*

1. **package** com.javatpoint;
2. **import** org.springframework.aop.ThrowsAdvice;
3. **public** **class** ThrowsAdvisor **implements** ThrowsAdvice{
4. **public** **void** afterThrowing(Exception ex){
5. System.out.println("additional concern if exception occurs");
6. }
7. }

Create the xml file as in the previous example, you need to change only the Validator class and advisor class.

*File: applicationContext.xml*

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="obj" **class**="com.javatpoint.Validator"></bean>
10. <bean id="ba" **class**="com.javatpoint.ThrowsAdvisor"></bean>
12. <bean id="proxy" **class**="org.springframework.aop.framework.ProxyFactoryBean">
13. <property name="target" ref="obj"></property>
14. <property name="interceptorNames">
15. <list>
16. <value>ba</value>
17. </list>
18. </property>
19. </bean>
21. </beans>

*File: Test.java*

1. **package** com.javatpoint;
3. **import** org.springframework.beans.factory.BeanFactory;
4. **import** org.springframework.beans.factory.xml.XmlBeanFactory;
5. **import** org.springframework.core.io.ClassPathResource;
6. **import** org.springframework.core.io.Resource;
8. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. Resource r=**new** ClassPathResource("applicationContext.xml");
11. BeanFactory factory=**new** XmlBeanFactory(r);
13. Validator v=factory.getBean("proxy",Validator.**class**);
14. **try**{
15. v.validate(12);
16. }**catch**(Exception e){e.printStackTrace();}
17. }
18. }

**Output**

1. java.lang.ArithmeticException: Not Valid Age
3. additional concern **if** exception occurs
5. at com.javatpoint.Validator.validate(Validator.java:7)
6. at com.javatpoint.Validator$$FastClassByCGLIB$$562915cf.invoke(<generated>)
7. at net.sf.cglib.proxy.MethodProxy.invoke(MethodProxy.java:191)
8. at org.springframework.aop.framework.Cglib2AopProxy$CglibMethodInvocation.invoke
9. Joinpoint(Cglib2AopProxy.java:692)
10. at org.springframework.aop.framework.ReflectiveMethodInvocation.
11. proceed(ReflectiveMethodInvocation.java:150)
12. at org.springframework.aop.framework.adapter.ThrowsAdviceInterceptor.
13. invoke(ThrowsAdviceInterceptor.java:124)
14. at org.springframework.aop.framework.ReflectiveMethodInvocation.
15. proceed(ReflectiveMethodInvocation.java:172)
16. at org.springframework.aop.framework.Cglib2AopProxy$DynamicAdvisedInterceptor.
17. intercept(Cglib2AopProxy.java:625)
18. at com.javatpoint.Validator$$EnhancerByCGLIB$$4230ed28.validate(<generated>)
19. at com.javatpoint.Test.main(Test.java:15)